# CWE Detail – CWE-125

## Description

The product reads data past the end, or before the beginning, of the intended buffer.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2023-1018: The reference implementation code for a Trusted Platform Module does not implement length checks on data, allowing for an attacker to read 2 bytes past the end of a buffer.

**•** CVE-2020-11899: Out-of-bounds read in IP stack used in embedded systems, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2014-0160: Chain: "Heartbleed" bug receives an inconsistent length parameter (CWE-130) enabling an out-of-bounds read (CWE-126), returning memory that could include private cryptographic keys and other sensitive data. (KEV)

**•** CVE-2021-40985: HTML conversion package has a buffer under-read, allowing a crash

**•** CVE-2018-10887: Chain: unexpected sign extension (CWE-194) leads to integer overflow (CWE-190), causing an out-of-bounds read (CWE-125)

**•** CVE-2009-2523: Chain: product does not handle when an input string is not NULL terminated (CWE-170), leading to buffer over-read (CWE-125) or heap-based buffer overflow (CWE-122).

**•** CVE-2018-16069: Chain: series of floating-point precision errors  
 (CWE-1339) in a web browser rendering engine causes out-of-bounds read  
 (CWE-125), giving access to cross-origin data

**•** CVE-2004-0112: out-of-bounds read due to improper length check

**•** CVE-2004-0183: packet with large number of specified elements cause out-of-bounds read.

**•** CVE-2004-0221: packet with large number of specified elements cause out-of-bounds read.

**•** CVE-2004-0184: out-of-bounds read, resultant from integer underflow

**•** CVE-2004-1940: large length value causes out-of-bounds read

**•** CVE-2004-0421: malformed image causes out-of-bounds read

**•** CVE-2008-4113: OS kernel trusts userland-supplied length value, allowing reading of sensitive information

## Related Attack Patterns (CAPEC)

* CAPEC-540

## Modes of Introduction

**•** Implementation: N/A

## Common Consequences

**•** Impact: Read Memory — Notes: An attacker could get secret values such as cryptographic keys, PII, memory addresses, or other information that could be used in additional attacks.

**•** Impact: Bypass Protection Mechanism — Notes: Out-of-bounds memory could contain memory addresses or other information that can be used to bypass ASLR and other protection mechanisms in order to improve the reliability of exploiting a separate weakness for code execution.

**•** Impact: DoS: Crash, Exit, or Restart — Notes: An attacker could cause a segmentation fault or crash by causing memory to be read outside of the bounds of the buffer. This is especially likely when the code reads a variable amount of data and assumes that a sentinel exists to stop the read operation, such as a NUL in a string.

**•** Impact: Varies by Context — Notes: The read operation could produce other undefined or unexpected results.

## Potential Mitigations

**•** Implementation: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. To reduce the likelihood of introducing an out-of-bounds read, ensure that you validate and ensure correct calculations for any length argument, buffer size calculation, or offset. Be especially careful of relying on a sentinel (i.e. special character such as NUL) in untrusted inputs. (Effectiveness: N/A)

**•** Architecture and Design: Use a language that provides appropriate memory abstractions. (Effectiveness: N/A)

## Applicable Platforms

**•** C (Class: None, Prevalence: Undetermined)

**•** C++ (Class: None, Prevalence: Undetermined)

## Demonstrative Examples

**•** However, this method only verifies that the given array index is less than the maximum length of the array but does not check for the minimum value (CWE-839). This will allow a negative value to be accepted as the input array index, which will result in a out of bounds read (CWE-125) and may allow access to sensitive memory. The input array index should be checked to verify that is within the maximum and minimum range required for the array (CWE-129). In this example the if statement should be modified to include a minimum range check, as shown below.